CSC 227 (Operating system)

Course project

(CPU Scheduler Simulation Report)

Student Name	ID
Abdullah Alqobaisi	444101556
Faisal Alqahtani	444101948
Abdullah Alyousef	444102160

Instructor: Dr. Mohammad R. Alsulmi

Software and Hardware Tools

Software Tools

Development Environment: Java

IDE: Visual Studio Code

Version Control: Git

System Calls Proposed

The simulation implements several system calls to mimic OS functionality:

`1.sysAllocMemory(MemoryManager memoryManager,int memoryRequired)`

- Arguments:
  - memoryManager: Reference to the memory management system
  - memoryRequired: Amount of memory (in MB) needed by the process
- Return Value: Boolean indicating whether memory was successfully allocated
- Functionality: Simulates memory allocation for a new process by checking available memory and reserving the required amount
- Implementation:

`2.sysFreeMemory(MemoryManager memoryManager, int memoryRequired)`

- Arguments:
  - memoryManager: Reference to the memory management system
  - memoryRequired: Amount of memory (in MB) to be freed
- Return Value: void
- Functionality: Releases memory back to the system when a process terminates

`3. sysChangeProcessState(PCB pcb, ProcessState newState)`

- Arguments:
  - pcb: Process Control Block of the target process
  - newState: New state to assign (NEW, READY, RUNNING, WAITING, TERMINATED)
- Return Value: void
- Functionality: Updates the process state during transitions through the process lifecycle

`4. sysProcessFinished(ProcessTracker tracker, PCB pcb)`

- Arguments:
  - tracker: ProcessTracker that monitors process completion
  - pcb: Process Control Block of the finished process
- Return Value: void
- Functionality: Records process completion and updates system metrics

Strengths and Weaknesses of the Program

Strengths:

1. **Multithreaded Design:** Uses separate threads for file reading, memory management, and process scheduling, reflecting real OS design principles.
2. **Memory Management:** Simulates memory allocation and deallocation, adding realism to the simulation.
3. **Starvation Prevention:** Implements aging in the priority scheduler to prevent process starvation.
4. **Comprehensive Metrics:** Tracks waiting time, turnaround time, and execution sequences for performance analysis.
5. **Visualized Output:** Presents results in both table format and Gantt chart representation.
6. **Modular Design:** Components are well-separated, allowing for easy extension or modification.

Weaknesses:

1. **Simplified Memory Model:** Uses a basic memory model with no paging, segmentation, or virtual memory capabilities.
2. **Limited I/O Operations:** No simulation of I/O operations or I/O waiting states for processes.
3. **Fixed Time Units:** Uses fixed time units rather than dynamic time calculations based on real CPU performance.
4. **No GUI:** Relies on console output rather than providing a graphical interface.
5. **Limited Process Interdependencies:** No implementation of process dependencies or interprocess communication.
6. **Single CPU Simulation:** Only simulates a single CPU core, without support for multi-core scheduling.

Multithreading Analysis

Thread Count

The application creates several threads:

1. Main Thread: Controls the simulation and user interaction
2. File Reading Thread: Reads process information from the input file
3. Memory Loading Thread: Handles memory allocation for processes
4. Scheduler Thread: Executes the selected scheduling algorithm (FCFS, Round Robin, or Priority)

In total, 4 threads are created for the basic operation of the simulator.

Performance Impact

Multithreading significantly improves the performance and realism of the application for several reasons:

1. **Parallel Operations:** Different system functions can operate concurrently, similar to a real OS. For example, while the scheduler is executing a process, the file reader can load new processes.
2. **Realistic Simulation:** Real operating systems handle multiple tasks concurrently, so multithreading provides a more accurate representation of OS behavior.
3. **Resource Utilization:** On multi-core systems, threads can execute in parallel on different cores, utilizing available hardware resources more effectively.
4. **Simulated Concurrency:** Allows for simulation of scenarios where processes arrive while others are being scheduled and executed.

Without multithreading, operations would have to be performed sequentially, resulting in an unrealistic simulation and poorer performance as the system would need to complete one task before starting another.

Output Visualization Preference

We prefer Gantt charts for CPU scheduler output because they:

1. Provide immediate visual representation of execution timeline
2. Clearly show process interleaving in algorithms like Round Robin
3. Make idle CPU time easily identifiable

Enable intuitive visual comparison between different scheduling algorithms

However, tables offer complementary benefits:

1. Present precise numerical data for quantitative analysis
2. Display comprehensive metrics in a compact format
3. Allow for sorting and filtering of process data

Our implementation uses both formats: Gantt charts for visual analysis and tables for detailed metrics, providing the best of both approaches for complete understanding of scheduler behavior.

Extending to a Complete OS Simulation

To turn our CPU scheduler into a full OS simulation, I would add:

1. **Memory Management**
    a. Add virtual memory with paging
2. **File System**
    a. Add basic file operations (read/write)
    b. Include file metadata and permissions
3. **I/O Handling**
    a. Add device queues for different I/O devices
    b. Implement I/O interrupts and handlers
    c. Create I/O wait states for processes
4. **Process Management**
    a. Add process creation/termination functions
    b. Include basic IPC mechanisms
    c. Implement semaphores for synchronization
5. **Multi-Core Support**
    a. Modify scheduler to work with multiple CPUs
    b. Add simple load balancing between cores

These additions would make the simulation much more realistic by showing how the CPU scheduler works with other OS components, giving us a better understanding of the whole system rather than just one part of it.